Université Nice Sophia Antipolis

Conception Orientée Objets

Programmation SOLID

Frédéric Mallet

http://deptinfo.unice.fr/~fmallet/

Université Nice Sophia Antipolis

Objectives

- Introduce some principles to guide the design
  - **S**ingle responsibility
  - **O**pen-Closed
  - (**L**iskov) Substitution
  - **I**nterface
  - **D**ependency inversion
- See Robert Martin's book
  - Agile Software Development: Principles, Patterns and Practices, Prentice Hall

Université Nice Sophia Antipolis

Single Responsibility Principle

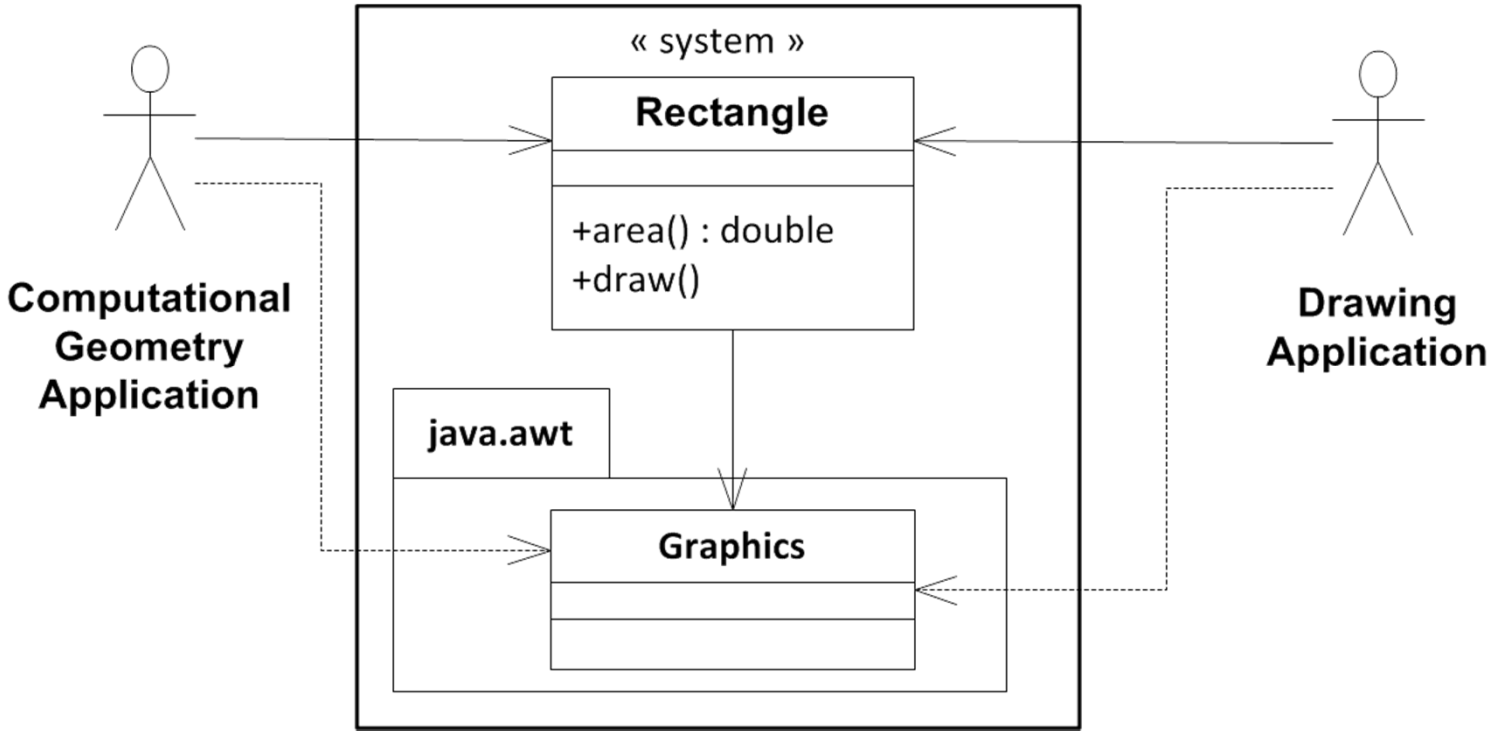
A class should have one, and only one, reason to change.

- Shared responsibilities means
  - Coupling the responsibilities:
    - Changes on one responsibility may impact the other one;
    - The need to rebuild, retest, repackage everything at each change.
  - Limit reusability;
  - Combining the dependencies.
- Warning: avoid needless complexity
  - Some responsibilities may actually be **intrinsically coupled**!

Université Nice Sophia Antipolis

SRP: One example of coupling

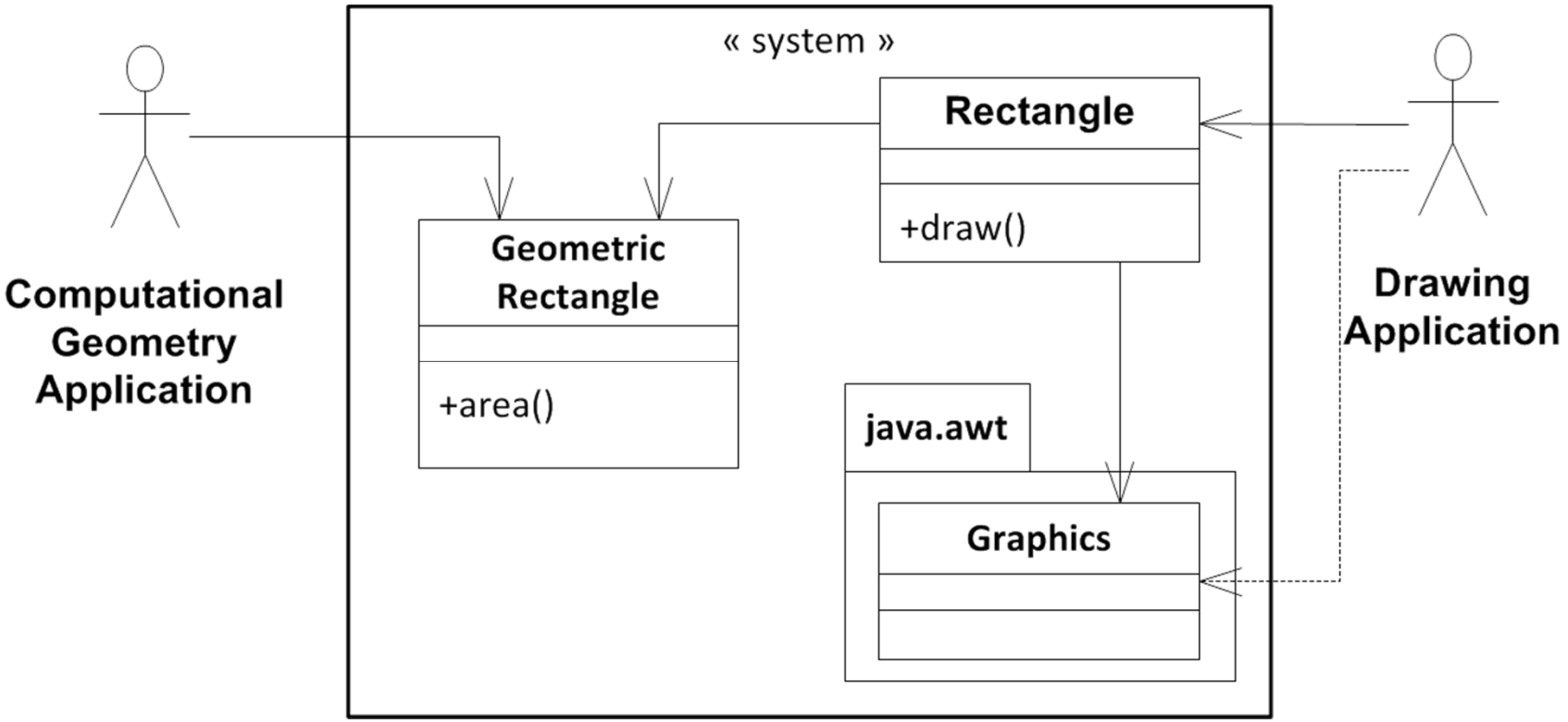
- A rectangle can be
  - A mathematical object for computational geometry;
  - A figure to be drawn in a GUI



Université Nice Sophia Antipolis

SRP: One example of decoupling

- A rectangle can be
  - A mathematical object for computational geometry;
  - A figure to be drawn in a GUI

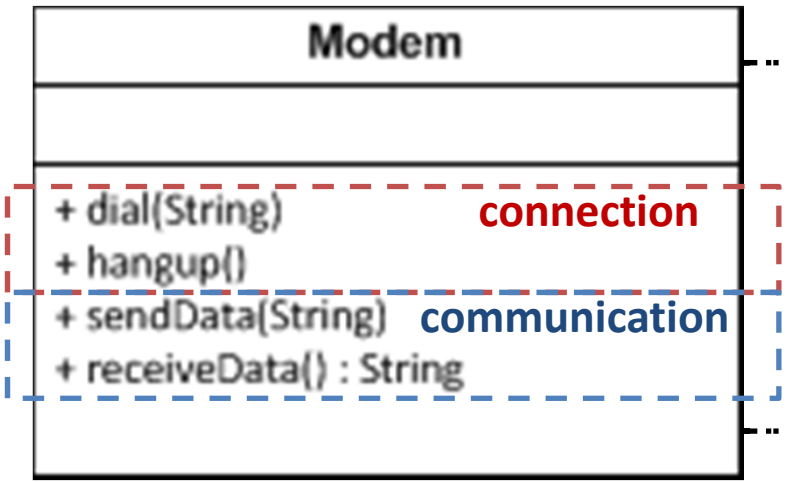


Université Nice Sophia Antipolis

SRP and Interfaces

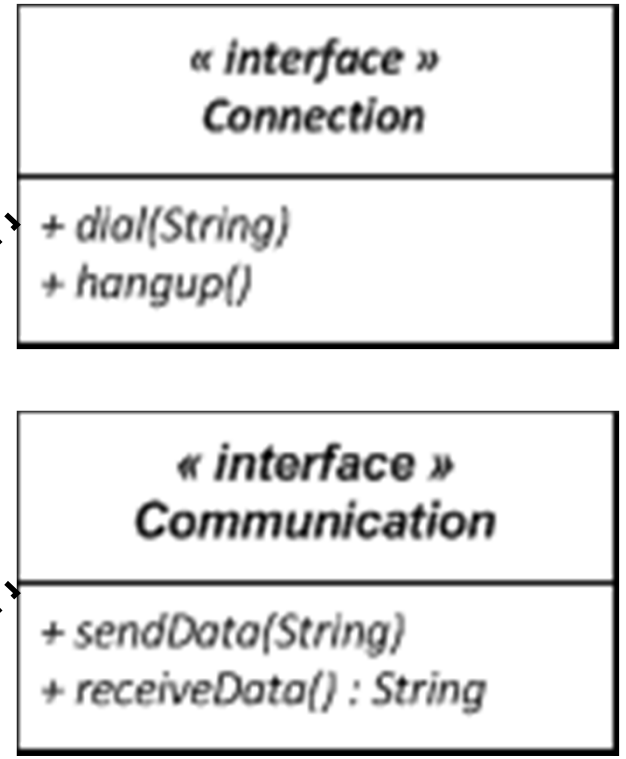
SRP violation

- A Modem
  - Handles connection
  - Send/receive information



Decoupled if necessary

- How to decouple?



Université Nice Sophia Antipolis

SRP: General comments

- Avoid coupling as much as possible
  - Testing is easier when coupling is minimal
- Two places where decoupling is essential
  - Graphical interface
    - Separate business model from the layout (e.g. MVC)
  - Persistence
    - Separate the business model from the technology to save/restore objects on disk

Université Nice Sophia Antipolis

Open-Closed

- Closed to modification
  - Encapsulation allows to control the modifications from the other classes
  - Want to limit the changes in the existing code
- Open for extensions
  - Requirements are meant to evolve all the time
  - Should easy allow extensions
    - By adding new classes, not modifying existing ones
    - The behavior of a module evolves without changing the module code (untouched .dll, .jar)

Université Nice Sophia Antipolis

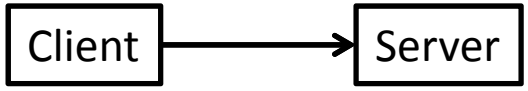
Open-Closed Principle

- Symptom
  - A single change results in a cascade of changes in dependent modules
    - Design Smell: Rigidity
- Diagnostic
  - Refactor your design to avoid such cascades
  - If OCP is applied well
    - add new code, not changing old code

Université Nice Sophia Antipolis

OCP: violated

- Simple example of violation

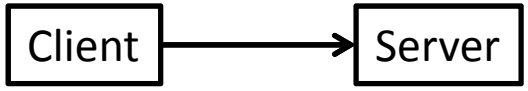


  - Any modification on the server impacts the client
    - At the very least needs to recompile

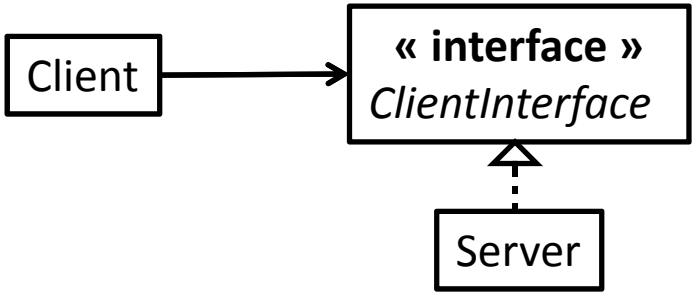
Université Nice Sophia Antipolis

OCP: fixing the violation

- Simple example of violation



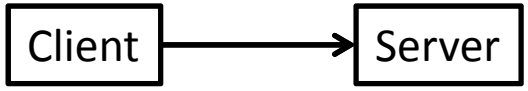
  - Any modification on the server impacts the client
    - At the very least needs to recompile
- Simple fix of the violation: *Strategy Pattern*
  - Introduce an abstract interface that serves the client



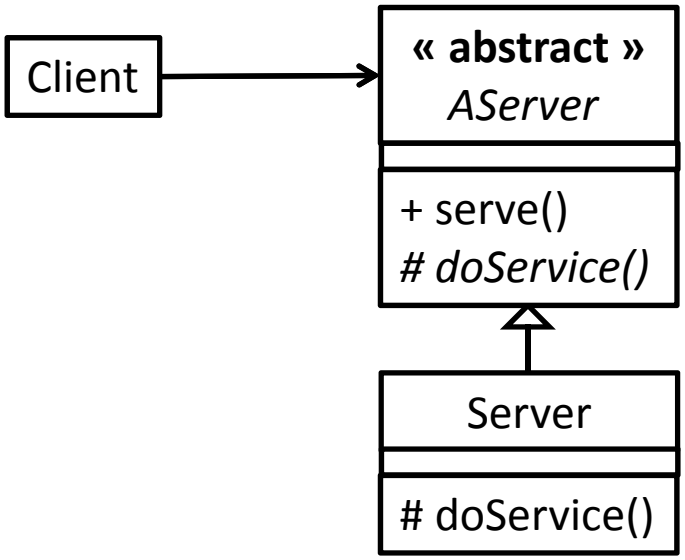
Université Nice Sophia Antipolis

OCP: fixing the violation

- Simple example of violation



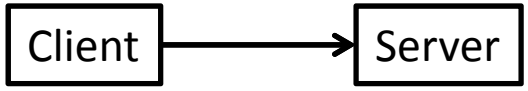
- Other fix of the violation: *Abstract Method Pattern*
  - Introduce an abstract method that can evolve



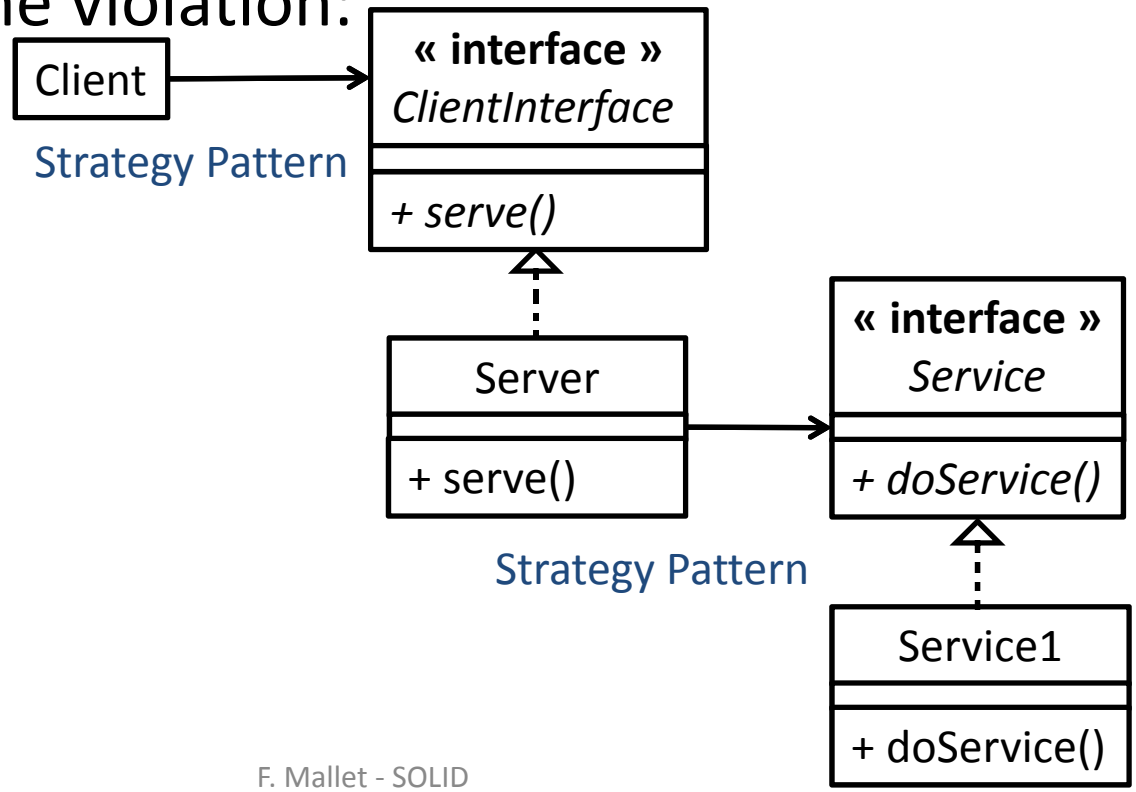
Université Nice Sophia Antipolis

OCP: fixing the violation

- Simple example of violation



- Other fix of the violation:



Université Nice Sophia Antipolis

OCP

- Where to stop?
  - It depends
- "Take the first bullet"
  - If you meet a case, where you cannot easily extend
  - Then you build a more general mechanism
- Beware of antipatterns
  - Bring *accidental complexity*

Université Nice Sophia Antipolis

Liskov substitution principle

- OCP relies on abstraction and polymorphism
- Inheritance is a way to achieve abstraction and polymorphism
- Liskov substitution principle (LSP)
  - Gives a rule to decide how to build a sound inheritance tree

Subtypes must be substitutable for their base types

Université Nice Sophia Antipolis

LSP

- Scenario Violation
  - Given a method f(b: BaseType)
  - Apply f to an object d of type DerivativeType such that DerivativeType inherits from BaseType
    - This causes f to fail
  - Fix f by testing the dynamic type of b
    - This causes an OCP violation => When creating other derivatives of BaseType, it is possible that f will also misbehave
    - Ex: If (b instanceof DerivativeType)
  - LSP violation has caused an OCP violation

Université Nice Sophia Antipolis

LSP: Examples

- Can we use inheritance to unify those classes?

Rectangle
- width : int - height : int
+ setWidth(int) + setHeight(int) + getWidth() : int + getHeight() : int + area()

Square
- side : int
+ setSide(int) + getSide() : int + area()

Line
- p1 : Point - p2 : Point
+ getSlope() : int + getIntercept() : Point + isOn(Point) : boolean

LineSegment
- p1 : Point - p2 : Point
+ getSlope() : int + getIntercept() : Point + isOn(Point) : boolean + getLength() : int

Université Nice Sophia Antipolis

LSP: Examples

- Can we use inheritance?
  - It depends (see Code)
  - Rectangle and Square
    - setWidth(int) implicitly assumes that it does not alter height
  - Line and LineSegment
    - Line assumes `line.isOn(line.getIntercept())`
    - May introduce a `LinearObject`

Université Nice Sophia Antipolis

LSP: general rules

- When to use inheritance?
  - When two classes share a common responsibility
    - Build a super class in charge of this responsibility
- Beware of pre and post-conditions when overriding
  - May only replace a precondition by one equal or weaker
  - May only replace a post-condition by one equal or stronger
    - Rectangle.setWidth:
      - post-condition: this.width == width && this.height == old.height
    - Square.setWidth:
      - post-condition: this.width == width [weaker]
- Should not (in general) remove functionality
  - Beware of degenerative functions [D extends B]
    - B.f() { /* some code */ }
    - D.f() { /* empty */ }

Université Nice Sophia Antipolis

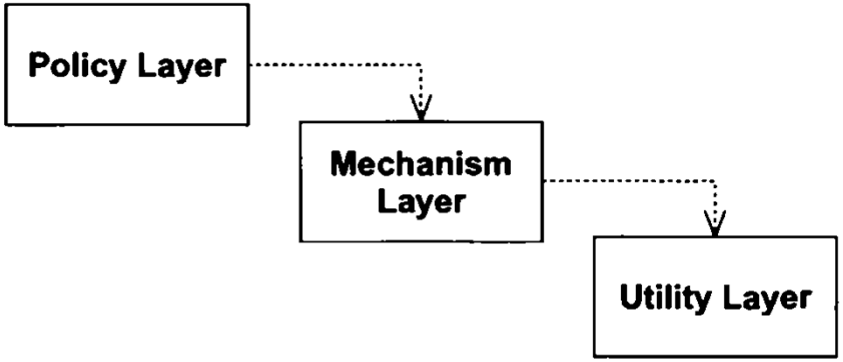
Dependency-Inversion Principle

- High-level modules should not depend on Low-Level Modules
  - This is contrary to usual functional decomposition
  - Both should depend on abstractions
- Abstractions should not depend on details
  - Details should depend on abstractions

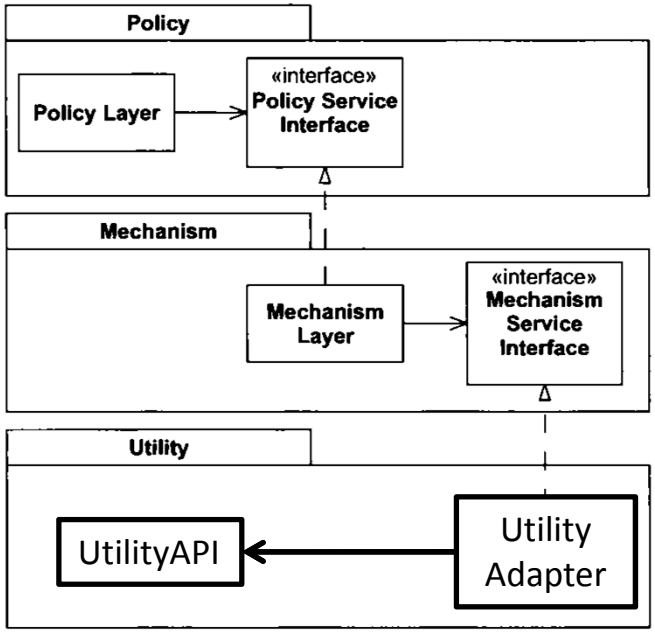
Université Nice Sophia Antipolis

Dependency-Inversion Principle

- DIP Broken



- DIP fixed



Adapter Pattern

Université Nice Sophia Antipolis

DIP: Example

- Copy

```
void copy() throws IOException {
    for (int c = System.in.read(); c != -1; c = System.in.read()) {
        System.out.write(c);
    }
}
```

- High-level concept depends on low-level implementation

Université Nice Sophia Antipolis

DIP: Example

- Copy: bad fix

```
FileOutputStream output;
boolean writeOnConsole;

void copy() throws IOException {
    for (int c = System.in.read(); c != -1; c = System.in.read()) {
        if (writeOnConsole) System.out.write(c);
        else output.write(c);
    }
}
```

- High-level concept depends on low-level implementation

Université Nice Sophia Antipolis

DIP: Example

- Copy: good fix

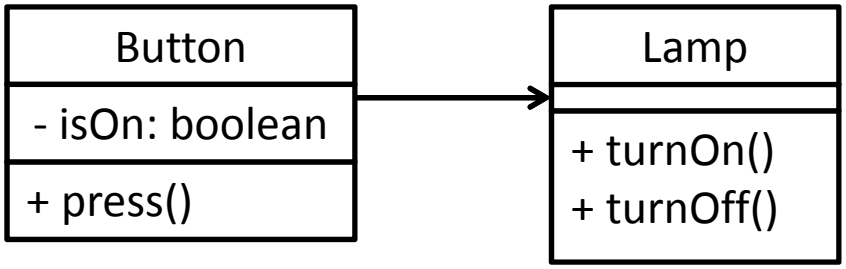
```
OutputStream output;

void copy() throws IOException {
    for (int c = System.in.read(); c != -1; c = System.in.read()) {
        output.write(c);
    }
}
```

- High-level concept depends on an abstraction

Université Nice Sophia Antipolis

DIP: Another example

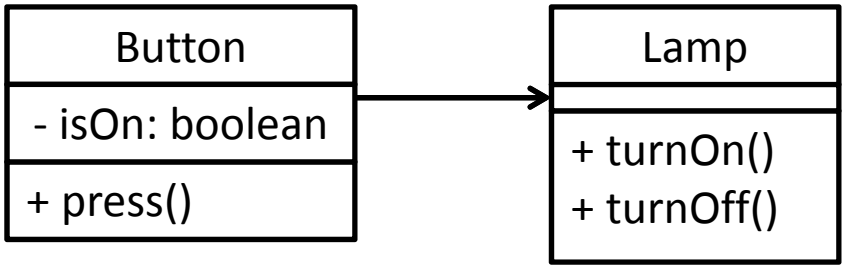
- Button and Lamp



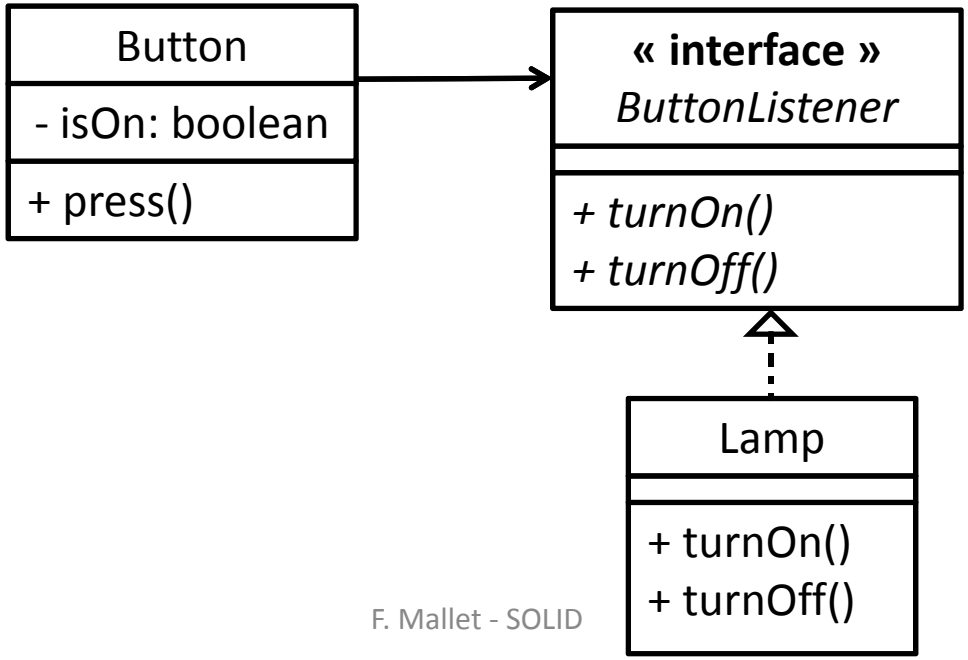
Université Nice Sophia Antipolis

DIP: Another example

- Button and Lamp



- One fix



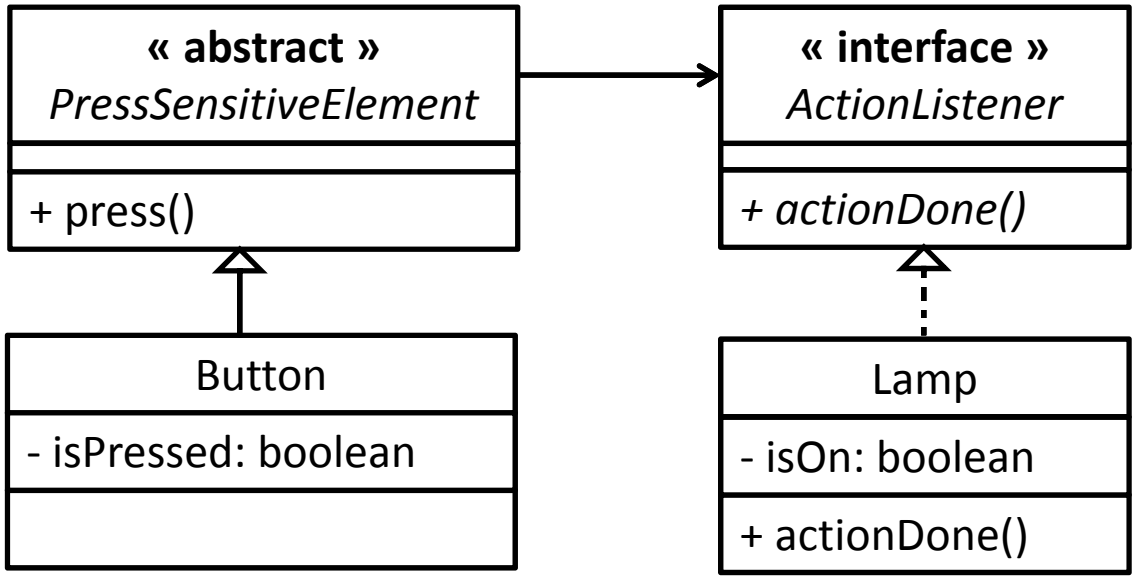
Listener pattern

Université Nice Sophia Antipolis

DIP: Another example

- Another abstraction

Listener pattern



Université Nice Sophia Antipolis

Interface Segregation Principle

Clients should not be forced to depend on methods that they do not use

- Symptom:
  - Some clients use one group of member methods
  - Other clients use another group
- Diagnostic: *"fat interfaces"*
- Cure
  - Split the interface into several sub-interfaces

Université Nice Sophia Antipolis

ISP: Example

- TimedDoor: Code

```
public class TimedDoor extends PlaybackListener
                implements ActionListener, Runnable {
        private boolean isOpened = false;
        private Timer timer = new Timer(100, this);
        private AdvancedPlayer player;

        void lock() { isOpened = false; timer.stop(); }
        void unlock() { isOpened = true; timer.setInitialDelay(5000); timer.setRepeats(false);
                timer.start();
        }

        boolean isOpen() { return isOpened; }

        void timeout() {
                ...
                this.player = new AdvancedPlayer( … );
                player.setPlayBackListener(this);
                Thread playerThread = new Thread(this,
                "AudioPlayerThread");
                playerThread.start();
                }
        }
        public void actionPerformed(ActionEvent arg0) {
                        timeout();
        }

        public void run() {
                        try {
                        player.play();
                        } catch (JavaLayerException e) {
                        e.printStackTrace();
                        }
        }
}
```

- Timed Door: Interface